# 分割統治法による多倍長演算の高速化 (数式処理における理論と応用の研究)

AUTHOR(S):

平山, 弘

CITATION:

平山, 弘. 分割統治法による多倍長演算の高速化 (数式処理における理論と応用の研究). 数理解析研究所講究録 2000, 1138: 247-255

ISSUE DATE:

2000-04

URL:

http://hdl.handle.net/2433/63809

数理解析研究所講究録
1138巻 2000年 247-255



# 分割統治法による多倍長演算の高速化

神奈川工科大学　平山　弘(Hiroshi Hirayama) *

## 1　はじめに

分割統治法 (Divided and Conquer) とは、大きな問題を小さな問題に分割し、高速化等を行う方法である。この方法を使う有名な計算法としてＦＦＴがある。ここでは、単純なトーナメント法を扱う。多倍長の計算を、小さな桁数の計算に分割し高速化をはかる方法である。

この方法は、相加相乗平均の原理を利用した円周率の高速計算法と異なり、10進数で1000桁以下の小さな桁数の数値の計算にも高速であり、さらに桁数が大きいときには、相加相乗平均の原理を利用した方法と同程度のオーダーの計算量で計算できる。また計算の原理が単純なので容易に利用できる。

以下では、単純な階乗の計算を例にとり、このトーナメント法の原理とその有効性を示す。また、右田等 [5] によって提案されたある種の Taylor 展開式の高速計算方法は、この計算方法を２行２列の行列の乗算の形に変形することによって、原理的には、階乗の計算と同じ方法であることを示す。

さらに、連分数の漸化式も２行２列の行列の乗算の形に変形することができることを示す。この式を利用すると、連分数もある種の Taylor 展開式と同様に効率的に計算できることを示す。

## 2　階乗の計算

簡単な例として、階乗の計算

$$n! = \prod_{k=1}^{n} k = 1 \cdot 2 \cdot 3 \cdots n \tag{1}$$

を考える。$n$ が小さい場合、前から順序よく計算しても計算時間に大きな違いが生じないので問題が生じない。$n$ が1000以上になると、前から順序よく計算する方法はあまり効率

*hirayama@sd.kanagawa-it.ac.jp

表 22: 階乗の計算時間（単位秒）

| n | 通常の計算 | 提案した計算法 |
|---|---|---|
| 1000 | 0.05 | 0.01 |
| 10000 | 2.583 | 0.151 |
| 20000 | 49.381 | 0.681 |

的な方法とはいえなくなる。$n$ が $2^m$ と表すことができる場合、2 個づつ組み合わせ

$$n! = (1 \cdot 2) \cdot (3 \cdot 4) \cdots ((n-1) \cdot n) \tag{2}$$

さらに、2 個組み合わせ

$$n! = ((1 \cdot 2) \cdot (3 \cdot 4)) \cdot ((5 \cdot 6) \cdot (7 \cdot 8)) \cdots (((n-3) \cdot (n-2)) \cdot ((n-1) \cdot n)) \tag{3}$$

のように次々と計算する。いわゆるトーナメント法で計算することができる。この方法で計算したものと、1 から順序良く計算した通常の方法で計算したときの計算時間を表 1 に示す。この計算は、Pentium II 450MHz を利用して計算した。10000 進数表現を使った自作の C++言語の多倍長演算プログラムを利用した。このプログラムでは、10 進数で 1000 桁以上の数値どうしの乗算には、高速フーリエ変換 (FFT) を利用して計算を行うようになっている。計算結果は、

$$1000! = 4.023872600770937735437024339 23 \times 10^{2567} \tag{4}$$

$$10000! = 2.846259680917054518906413212 11 \times 10^{35659} \tag{5}$$

$$20000! = 1.819206320230345134827641756 86 \times 10^{77337} \tag{6}$$

となる。

表 1 から、高速フーリエ変換 (FFT) があまり有効に働かない 1000! でも 5 倍程度高速に計算できることがわかる。FFT が有効に働く 10000! や 20000! の計算では、それぞれ、17.1 倍、72.5 倍と非常に高速になる。

## 3 階乗計算量の評価

$n!$ の計算量の評価を行う。簡単化のために、$n = 2^m$ とし、掛算する数値の桁数はすべて $n$ と同じ桁数 $K$ と仮定する。計算する項数は $n$ となる。

前提となる条件は、次のようなものである。$N$ 桁の数の乗算の計算量を $M(N)$ とする。$M(N)$ は、通常の計算方法では $M(N) = O(N^2)$ である。$N$ が非常に大きな桁数の数のとき、高速フーリエ変換が利用できて、$M(N) = O(N(\log N)^2)$ で計算できることが知られている。

最初の段階では、$\frac{n}{2}$ 回の $K$ 桁の数値の乗算を必要となるので、計算量は $\frac{Cn}{2}M(K)$ となる（ここで $C$ は定数)。次の段階では、項数は $\frac{n}{4}$ となり、乗算は各項について必要なので、計算量は、$\frac{Cn}{4}M(2K)$ となる。次の段階では、計算桁数は倍、計算項数は半分になるので、$\frac{Cn}{8}M(4K)$ となる。従って、総計算量 $TM$ は、

$$TM = Cn\frac{1}{2}M(K) + \frac{1}{4}M(2K) + \frac{1}{8}M(4K) + \cdots + \frac{1}{n}M(\frac{n}{2}K) \tag{7}$$

となる。$m$ が大きいとき、$M(m) = O(m(\log m)^2)$ であることを考慮すると

$$TM \approx Cn(\log n)\frac{1}{n}M(\frac{n}{2}K) \approx Cn(\log n)^3 \tag{8}$$

が得られる。

## 4 ある種の級数の計算

右田等 [5] によって、次の (9) の形式で表現できる級数は、係数の桁数が小さいく、計算精度を p が十分大きいとき、の計算量で計算することができることが示されている。この計算法（縮約法）は、級数 $S$ が、次の形で表現できるとき、すなわち

$$S = \frac{1}{C_0}(A_0 + \frac{B_0}{C_1}(A_1 + \frac{B_1}{C_2}(A_2 + \frac{B_2}{C_3}(A_3 + \frac{B_3}{C_4}(A_4 + \cdots + \tag{9}$$

という形式で表されるとき、次のように変形することができる。

$$S = \frac{1}{C_0}(A_0 + \frac{B_0}{C_1C_2}(A_1C_2 + A_2B_1 + \frac{B_1B_2}{C_3}(A_3 + \frac{B_3}{C_4}(A_4 + \cdots + \tag{10}$$

これらの計算は、すべて、分数のままで計算すると、各数値は短い桁数の数値であるから、階乗の計算と同じように高速に計算できる。計算する数値の桁数が大きくなった場合、高速フーリエ変換を利用した乗算法を使うと、階乗と同じく $O(n(\log n)^3)$ の計算量で計算できる。

また、(9) の式の $n$ 項までの和は、

$$\begin{pmatrix} Q_n & P_n \\ 0 & R_n \end{pmatrix} = \prod_{k=0}^{n} \begin{pmatrix} B_k & A_k \\ 0 & C_k \end{pmatrix}$$

$$= \begin{pmatrix} B_0 & A_0 \\ 0 & C_0 \end{pmatrix} \begin{pmatrix} B_1 & A_1 \\ 0 & C_1 \end{pmatrix} \begin{pmatrix} B_2 & A_2 \\ 0 & C_2 \end{pmatrix} \cdots \begin{pmatrix} B_n & A_n \\ 0 & C_n \end{pmatrix} \tag{11}$$

を第 $n$ 項まで計算し、$\frac{P_n}{R_n}$ を計算することに相当する。最後の除算も Newton 法を利用すると乗算と同じオーダーで計算できることが知られているので、計算量のオーダーは、階乗の計算量と同じになる。[2]

上の式を見ればわかるように、行列の要素の 1 個は常に 0 になる。このため、ゼロの要素がない場合、要素間の乗算回数が 8 に対し、4 回と半分の回数で計算できる。

## 5 行列の乗算形式にできる級数の具体例

非常に多くの級数を行列の乗算形式で表現できる。指数関数は次のように表現できる。

$$e^x = \frac{1}{1}(1 + \frac{x}{1}(1 + \frac{x}{2}(1 + \frac{x}{3}(1 + \frac{x}{4}(1 + \cdots (1 + \frac{x}{n}(1 + \cdots \tag{12}$$

これは、次のような行列の乗算形式に変形できる。

$$\begin{pmatrix} Q_n & P_n \\ 0 & R_n \end{pmatrix} = \begin{pmatrix} x & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x & 1 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} x & 1 \\ 0 & 3 \end{pmatrix} \cdots \begin{pmatrix} x & 1 \\ 0 & n \end{pmatrix} \tag{13}$$

対数関数は、

$$\frac{\log x}{x} = \frac{1}{1}(1 + \frac{x}{2}(1 - \frac{2x}{3}(1 - \frac{3x}{4}(1 - \frac{4x}{5}(1 - \cdots (1 - \frac{(n-1)x}{n}(1 - \cdots \tag{14}$$

$$\begin{pmatrix} Q_n & P_n \\ 0 & R_n \end{pmatrix} = \begin{pmatrix} -x & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -2x & 1 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} -3x & 1 \\ 0 & 3 \end{pmatrix} \cdots \begin{pmatrix} -nx & 1 \\ 0 & n \end{pmatrix} \tag{15}$$

このほかに三角関数、逆三角関数、双曲線関数など簡単な規則で各項の係数が表現できる多くの関数がこの行列の乗算形式に変形できる。Taylor 展開の係数が簡単な規則で表現できない $\tan x$ などが例外的に表現できないだけである。

## 6 2 進 10 進数の相互変換

現在の多くの計算機が内部表現として 2 進数を利用している。このため、多倍長数値表現として 2 進数表示を使うと、メモリー効率だけでなく計算速度もかなり速くなる。

一部インライン・アセンブラーを使用した自作したプログラム [3][4] では、10 進数で約 1000 桁の数値の計算で、およそ 5 倍の速度向上があった。高速フーリエ変換による乗算は、10 進数をベースとした多倍長演算では、およそ 1000 桁から 2000 桁で通常の乗算法より高速になるが、2 進数をベースにした演算ではおよそ 10000 桁以上の数値でなければ通常の乗算が高速であった。

このように、よく使われる可能性のある低精度で速度向上がみられるので 2 進数表現は、多倍長演算は大変魅力的である。しかしながら、多倍長数を 2 進数表現にするためには、アセンブラーを使う必要があり、機械に依存したプログラムになるだけでなく、最終結果を 10 進数に変換するとき、これまでよく知られた方法では、変換する数値の桁数を $p$ とすると、計算量が $O(p^2)$ であり、非常に時間がかかるという問題が生じる。特に高速化の進んだ円周率の計算では、大きな欠点となる。変換する数値の桁数が 10 進数でおよそ 100 桁程度の短い場合には、従来の方法が有効であるが、10 万桁とか 100 万桁の数値になるとその変換に非常に多くの時間がかかる。

ここでは、上の級数の計算法が、2 進数 10 進数の相互変換の有力な計算法になることを示す。

実数 $a$ を 2 進数で表現すると、

$$a = \sum_{k=0}^{\infty} 2^{N-k} \tag{16}$$

と表現できる。これは、次のように (9) の形式に変形できる。

$$\begin{aligned} a &= a_0 2^N + a_1 2^{N-1} + a_2 2^{N-2} + \cdots \\ &= 2^N (a_0 + \tfrac{1}{2}(a_1 + \tfrac{1}{2}(a_2 + \tfrac{1}{2}(a_3 + \cdots \end{aligned} \tag{17}$$

この式を、基数 $r$ の式に一般化すると、

$$a = \sum_{k=0}^{\infty} r^{N-k} \tag{18}$$

となります。(9) の形式に変形すると

$$\begin{aligned} a &= a_0 r^N + a_1 r^{N-1} + a_2 r^{N-2} + \cdots \\ &= r^N (a_0 + \tfrac{1}{r}(a_1 + \tfrac{1}{r}(a_2 + \tfrac{1}{r}(a_3 + \cdots \end{aligned} \tag{19}$$

この計算を、10 進数表現の多倍長演算プログラムで計算すれば、10 進数に変換できる。この時の計算量は、精度を $p$ とすれば $O(p(\log p)^3)$ の計算量で計算できる。実際の計算では、(19) の基数 $r$ は、$2^{16}$ とか $2^{32}$ になります。

(19) の級数部分を行列の乗算形式で書くと、

$$\begin{pmatrix} 1 & P_n \\ 0 & R_n \end{pmatrix} = \begin{pmatrix} 1 & a_0 \\ 0 & r \end{pmatrix} \begin{pmatrix} 1 & a_1 \\ 0 & r \end{pmatrix} \begin{pmatrix} 1 & a_2 \\ 0 & r \end{pmatrix} \cdots \begin{pmatrix} 1 & a_n \\ 0 & r \end{pmatrix} \tag{20}$$

となる。この式を計算し、$a = r^N \frac{P_n}{R_n}$ として、$a$ が求められる。

逆に、基数を 10 にして、2 進数表現を使う多倍長演算プログラムで計算すれば、2 進数に変換することになります。この時の計算量も、2 進数 10 進数の変換と同様に、$O(p(\log p)^3)$ の計算量で計算できる。

計算のオーダーは、(9) の級数と同じであるが、(20) の式からわかるように、1 回の行列の乗算は、2 回の要素間の乗算しか必要としない。このため、通常のこれらの計算より高速である。

## 7 連分数の計算法

連分数は、次のように定義する。

$$\frac{P_n}{Q_n} = b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{\ddots + \cfrac{a_n}{b_n}}}} = b_0 + \frac{a_1}{b_1} + \frac{a_2}{b_2} + \cdots + \frac{a_n}{b_n} \tag{21}$$

このとき、よく知られているように、次の漸化式 [1] が成り立つ。

$$\begin{aligned} P_n &= b_n P_{n-1} + a_n P_{n-2} \\ Q_n &= b_n Q_{n-1} + a_n Q_{n-2} \end{aligned} \quad (n = 1, 2, \cdots) \tag{22}$$

ただし、$P_0 = b_0$、$Q_0 = 1$、$P_{-1} = 1$、$Q_{-1} = 0$ とする。この式は、次のように行列の乗算の形式で書くことができる。

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} P_{n-1} & P_{n-2} \\ Q_{n-1} & Q_{n-2} \end{pmatrix} \begin{pmatrix} b_n & 1 \\ a_n & 0 \end{pmatrix} \tag{23}$$

この関係式を利用すると、(21) の関係式は、

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} b_1 & 1 \\ a_1 & 0 \end{pmatrix} \begin{pmatrix} b_2 & 1 \\ a_2 & 0 \end{pmatrix} \cdots \begin{pmatrix} b_n & 1 \\ a_n & r \end{pmatrix} \tag{24}$$

を計算し、$\frac{P_n}{Q_n}$ を計算すれば、連分数の値を計算できる。このように変形できれば、階乗の計算と同様に高速に計算できる。

式は、級数の場合と似ているが、級数の場合、行列の要素の 1 つが常に 0 であるが、連分数の場合、0 にはならない。このため、連分数の方が、計算時間を多く必要とする。

## 8 行列の乗算形式にできる連分数の具体例

連分数表現は、次の例のように、Taylor 展開と同様に簡単に表現関数がある。

## 8.1 逆正接関数

逆正接関数は、次のように連分数展開できる。

$$\tan^{-1} = \frac{x}{1} + \frac{x^2}{3} + \frac{4x^2}{5} + \frac{9x^2}{7} + \cdots + \frac{n^2x^2}{2n+1} + \cdots \tag{25}$$

この式は、次のように、行列の乗算の形式で表現できる。

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x^2 & 1 \\ 3 & 0 \end{pmatrix} \cdots \begin{pmatrix} n^2x^2 & 1 \\ 2n+1 & 0 \end{pmatrix} \tag{26}$$

この式を使って円周率を 10 万桁まで計算するために、$x = 1$ を代入し、130626 項まで計算した。96.513 秒で計算することができた。(Pentium II 450MHz 使用) $\tan^{-1} 1$ を計算するために、$\tan^{-1} x$ の Taylor 展開式に $x = 1$ を代入することはほとんど考えられないことである。理論的には、収束し計算できるが、その収束は非常に遅く、実際上収束しない級数であるからである。

この非常に収束が遅い級数に対応する連分数は、実用に耐える程度の速さで収束し計算値を求めることができる。

従来から円周率の計算に利用されてきた Machin の公式

$$\pi = 16 \tan^{-1} \frac{1}{5} - 4 \tan^{-1} \frac{1}{239} \tag{27}$$

を用いれば、さらに高速に計算できる。1 万桁で 1.4 秒、10 万桁で 20.7 秒、20 万桁で 94.9 秒、50 万桁で 371.0 秒、100 万桁で 1374.6 秒であった。

この計算時間は、メモリー管理を行っていない多倍長演算ルーチンを利用した場合の結果である。メモリー管理をしないでも、200M バイト程度のメモリーを持っているコンピュータでは、10 万桁程度までの大きさのメモリー程度ならば、あまり問題とならないが 100 万桁程度となるとかなり影響する。

これらの計算の中の行列の乗算では、行列の要素を何回も使って乗算を行う。この場合、これらの数値のフーリエ変換は 1 度行えば十分であるが、今回のこの時間計測では乗算回数だけフーリエ変換を行っている。

## 8.2 対数関数

対数関数も次のような形に連分数展開できる。

$$\log\left(\frac{1+x}{1-x}\right) = \frac{2x}{1} - \frac{x^2}{3} - \frac{4x^2}{5} - \frac{9x^2}{7} - \cdots - \frac{n^2x^2}{2n+1} - \cdots \tag{28}$$

この式も次のように行列の乗算の形に変形できる。

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 2x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} -x^2 & 1 \\ 3 & 0 \end{pmatrix} \cdots \begin{pmatrix} -n^2x^2 & 1 \\ 2n+1 & 0 \end{pmatrix} \tag{29}$$

この式に $x = \frac{1}{3}$ を代入すると、$\log 2$ の値を計算できる。この方法で計算すると 1 万桁を 0.77 秒で計算することができる。

## 8.3 その他の基本関数の連分数展開式

指数関数 $e^x$ などが簡単な係数を持つ連分数展開できるが、三角関数の $\sin x$ や $\cos x$ が簡単な係数を持つ連分数展開できない。

Taylor 展開では、複雑な係数を持つ正接関数は、次のように簡単な係数をもつ連分数展開式に変換できるものもある。

$$\tan x = \frac{x}{1} - \frac{x^2}{3} - \frac{x^2}{5} - \frac{x^2}{7} - \cdots - \frac{x^2}{2n+1} - \cdots \tag{30}$$

# 9 特別な連分数

平方根は、巡回する連分数で表現できる。例えば

$$\sqrt{2} = 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \cdots \tag{31}$$

と表現できる。これは、

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix} \cdots \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix} \tag{32}$$

すなわち、

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}^n \tag{33}$$

となる。これは、行列のべき乗の計算であり、数値のべき乗と同様に効率的に計算できる。最初の一部を除いて、巡回する部分を掛け算すれば、すべて同じ行列の乗算となる。このような場合、さらに高速な計算が可能である。この計算法で計算すると、1 万桁の計算で 0.31 秒であった

## 10 連分数形式の円周率の公式

連分数で円周率を Taylor 級数を使う方法より効率的に求める公式は、知れてないが、Wallis 公式の拡張である

$$\pi = \frac{1}{n}\left(\frac{2\cdot 4\cdot 6\cdots(2n)}{1\cdot 3\cdot 5\cdots(2n-1)}\right)^2 \left(1+\frac{2}{8n-1}+\frac{1\cdot 3}{8n}+\frac{3\cdot 5}{8n}+\frac{5\cdot 7}{8n}+\cdots\right) \quad (34)$$

が知られている。10 万桁以上の計算では、Machin の公式より、いくらか高速であった。この場合、$n$ を計算したい桁数と同じ位の数値にする必要がある。たとえば、10 万桁計算するならば、$n = 100000$ とする必要がある。階乗の部分は、前で示したトーナメント法で計算する必要がある。

## 11 まとめ

桁数の小さい数値の乗算は、トーナメント法によって効率的に計算できる。数が非常に大きくなった場合、高速フーリエ変換を利用した方法で高速に計算できるので、乗算する数値の個数を $n$ とすると $O(n(\log n)^3)$ の計算量で計算できる。

ある種の級数と連分数の計算アルゴリズムが行列の積に表現できることを示した。その行列の乗算をトーナメント法によって高速に計算できることを示した。これにより、多くの関数値を高速に計算できる。

その重要な応用例として、2 進 10 進数間の相互変換の計算量も $O(p(\log p)^3)$ となり高速計算できることを示した。

## 12 参考文献

[1] M. Abramowitz and I. A. Stegun: Handbook of Mathmatical Function, Dover, 1965

[2] R.P.Brent : A Fortran Multiple-Precision Arithmetic Package, *ACM Trans. Math. Soft.*, 4, 1978,57–70

[3] 平山 弘 : 多倍長計算プログラムパッケージ MPPACK, 東京大学大型計算機センター, 1992

[4] 平山 弘 : C++言語による高精度計算パッケージの開発, 日本応用数理学会, 5(3), 1995, 123–134

[5] 右田, 天野, 浅田, 藤野: 級数の集約による多倍長数の計算法と$\pi$の計算への応用, 情報処理学会研究報告,98(74), 1998, 31–36, 1998